SKIP A HEARTBEAT: OPENSSL HEARTBLEED VULNERABILITY & PREDICTION OF EXPLOITATION

BASED ON CVSS USING NAIVE BAYES ALGORITHM

Mehak Bashir

SKIP A HEARTBEAT: OPENSSL HEARTBLEED VULNERABILITY & PREDICTION OF EXPLOITATION BASED ON CVSS USING NAIVE BAYES ALGORITHM

Er. Mehak Bashir

MASTER OF TECHNOLOGY

IN

COMPUTER SCIENCE ENGINEERING

Under the Supervision of:

Dr. MUHEET AHMED BUTT (Scientist 'D', Kashmir University)

Dr. MAJID ZAMAN BABA (Scientist 'D', Kashmir University)

Prof. HARKESH SEHRAWAT (Professor, Maharishi Dayanand University)

ACKNOWLEDGEMENTS

This book would not have been possible without the input and support provided by several people. The author would like to express a deep sense of gratitude and special thanks to Dr. Muheet Butt, Scientist 'D', Kashmir University. It would have been impossible to complete this work in this manner without his wise counsel and able guidance.

The constant guidance and encouragement received from Dr. Majid zaman, Scientist 'D', Kashmir University, who has been very helpful in completing the book, for which the author is highly indebted to his good-self.

The author expresses profound gratitude to Prof. Harkesh Sehrawat, Professor, Maharishi Dayanand University, for his intellectual support throughout the course of this work.

Finally, the author would like to express thanks to the many silent professionals whose work, advice, and feedback influenced my thinking and work on this book. Despite the efforts of so many to make this report better, there are undoubtedly weaknesses and errors that remain. These are entirely the fault of the author.

Mehak Bashir

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The Open Secure Sockets Layer (OpenSSL) provides secure platform for transactions, such as online shopping, online banking and emails etc., that take place over/across the internet. It is widely used open source implementation of the Secure Sockets Layer (SSL) and Transport Layer Security (TLS). Vulnerabilities have however been found in the OpenSSL which has resulted in a wide public outcry all over the world. A confounding computer bug called “Heartbleed” is causing major security worries across the internet. Heartbleed affects many things, including web servers, routers that connect office networks to the internet, mobile apps and VPNs (Virtual Private Network). It has been estimated that 60 percent of secure web sites that are using OpenSSL are affected. In addition, Heartbleed cannot be traced. The Heartbleed Bug has sent shockwaves all over the internet. Not only has all of this user data been directly compromised, but, what are worse, the private keys of the servers running the vulnerable versions of OpenSSL were also almost certainly compromised. Patching of affected applications or/and upgrade to versions that are not vulnerable, is recommended/suggested, in order to mitigate the risks identified.

The thesis/work describes OpenSSL Heartbleed vulnerability and also proposes a methodology that explains the severity of exploitation posed by some common types of vulnerabilities, based on Common Vulnerability Scoring System (CVSS), using Naive Bayes classification algorithm.

CHAPTER 1

INTRODUCTION

1.1 OpenSSL Heartbleed

The OpenSSL is an open source implementation of the Secure Sockets Layer (SSL) and the Transport Layer Security (TLS) [7].The OpenSSL platform provides security when data is transferred from one point of the internet to another part [1]. The Secure socket layer (SSL) is the most popular protocol used on the Internet for secure transfer of data [4]. The OpenSSL protocol is used in two-thirds of all websites to prevent hackers from stealing sensitive information like passwords or credit card data [5]. If the data being transferred is edited/changed/ updated along the way, data integrity is compromised and if the data is accessed and falls into the wrong hands, confidentiality of data is lost. Data Integrity and confidentiality should be maintained as data moves from point to point. The OpenSSL protocol works by authenticating the server to the client and client to server through the use of digital certificates signed by a trusted third party. Private and public keys are also used in the OpenSSL to provide security. The OpenSSL protocol is however subject to vulnerabilities [2], [3] whether directly or indirectly. This can be seen by the trusted third parties who authenticate the identities of transacting individuals have been exposed to continuous attacks/threats. [6]. various other vulnerabilities have been found

within the OpenSSL protocol and the most notable has been the Heartbleed bug.

The name 'Heartbleed' itself explains the vulnerability – 'Heart' of the Heartbleed came from Heartbeat protocol and 'bleed' stands for data leakage. That means data leakage in the Heartbeat protocol implementation, specifically the OpenSSL implementation of the protocol.

1.2 Naive Bayes Classifier

Naive Bayes is a kind of classifier which uses the Bayes Theorem. It predicts membership probabilities for each class such as the probability that given record or data point belongs to a particular class. The class with the highest probability is considered as the most likely class. This is also known as **Maximum A Posteriori (MAP)**.

The MAP for a hypothesis is:

$$\begin{aligned} \textbf{MAP(H)} &= \max(P(H|E)) \\ &= \max((P(E|H)*P(H))/P(E)) \\ &= \max(P(E|H)*P(H)) \end{aligned}$$

P (E) is evidence probability, and it is used to normalize the result. It remains same so, removing it won't affect.

Naive Bayes is a classification algorithm for binary (two-class) and multi-class classification problems. The technique is easiest to understand when described using binary or categorical input values [8].

1.3 Vulnerability

Vulnerability, in information technology (IT), is a flaw in code or design that creates a potential point of security compromise for an endpoint or network.

Vulnerabilities create possible attack vectors, through which an intruder could run code or access a target system's memory. The means by which vulnerabilities are exploited are varied and include code injection and buffer overruns; they may be conducted through hacking scripts, applications and free hand coding.

Vulnerabilities are constantly being researched and detected by the security industry, software companies, cybercriminals and other individuals. Some companies offer bug bounties for these discoveries. Nevertheless, when vulnerability disclosure is considered, the question of how much information to provide and when to make it public is a contentious issue.

Some people argue for full and immediate disclosure, including the specific information that could be used to exploit the vulnerability; others believe that vulnerability information should not be published at all because the information can be used by an intruder. A zero-day exploit, for example, takes place as soon as vulnerability becomes generally known. To mitigate risk, many experts believe that limited information should be made available to a selected group after some specified amount of time has elapsed since detection.

Both black hats and white hats regularly search for vulnerabilities and test exploits, however, and if a cybercriminal finds a useful and

unreported security hole, he is likely to take advantage of it. Proponents of disclosure maintain that it leads to more patching of vulnerabilities and more secure software [2].

1.4 Types of Security Vulnerabilities

Most software security vulnerabilities fall into one of a small set of categories:

- buffer overflows
- unvalidated input
- race conditions
- access-control problems
- weaknesses in authentication, authorization, or cryptographic practices

1.4.1 Buffer Overflows

A buffer overflow occurs when an application attempts to write data past the end (or, occasionally, past the beginning) of a buffer.

Buffer overflows can cause applications to crash, can compromise data, and can provide an attack vector for further privilege escalation to compromise the system on which the application is running.

Books on software security invariably mention buffer overflows as a major source of vulnerabilities. Exact numbers are hard to come by, but as an indication, approximately 20% of the published exploits reported by the United States Computer Emergency Readiness Team (US-CERT) for 2004 involved buffer overflows.

Any application or system software that takes input from the user, from a file, or from the network has to store that input, at least

temporarily. Except in special cases, most application memory is stored in one of two places:

- *stack*— A part of an application's address space that stores data that is specific to a single call to a particular function, method, block, or other equivalent construct.
- *heap*— General purpose storage for an application. Data stored in the heap remains available as long as the application is running (or until the application explicitly tells the operating system that it no longer needs that data).

Class instances, data allocated with malloc, core foundation objects, and most other application data resides on the heap. (Note, however, that the local variables that actually point to the data are stored in the stack.)

Buffer overflow attacks generally occur by compromising the stack, the heap, or both [1].

1.4.2 Unvalidated Input

As a general rule, we should check all input received by our program to make sure that the data is reasonable.

For example, a graphics file can reasonably contain an image that is 200 by 300 pixels, but cannot reasonably contain an image that is 200 by -1 pixel. Nothing prevents a file from claiming to contain such an image, however (apart from convention and common sense). A naive program attempting to read such a file would attempt to allocate a buffer of an incorrect size, leading to the potential for a heap overflow attack or other problem. For this reason, we must check our input data carefully. This process is commonly known as input validation or sanity checking.

Any input received by our program from an untrusted source is a potential target for attack. (In this context, an ordinary user is an untrusted source.) Examples of input from an untrusted source include (but are not restricted to):

- *text input fields*
- *commands passed through a URL used to launch the program*
- *audio, video, or graphics files provided by users or other processes and read by the program*
- *command line input*
- *any data read from an untrusted server over a network*
- *any untrusted data read from a trusted server over a network (user-submitted HTML or photos on a bulletin board, for example)*

Hackers look at every source of input to the program and attempt to pass in malformed data of every type they can imagine. If the program crashes or otherwise misbehaves, the hacker then tries to find a way to exploit the problem. Unvalidated-input exploits have been used to take control of operating systems, steal data, corrupt users' disks, and more. One such exploit was even used to "jail break" iPhones [1].

Validating Input and Inter-process Communication describes common types of input-validation vulnerabilities and what to do about them.

1.4.3 Race Conditions

A race condition exists when changes to the order of two or more events can cause a change in behavior. If the correct order of execution is required for the proper functioning of the program, this is a bug. If an attacker can take advantage of the situation to insert malicious code,

change a filename, or otherwise interfere with the normal operation of the program, the race condition is security vulnerability. Attackers can sometimes take advantage of small time gaps in the processing of code to interfere with the sequence of operations, which they then exploit [2].

1.4.4 Inter-process Communication

Separate processes—either within a single program or in two different programs—sometimes have to share information. Common methods include using shared memory or using some messaging protocol, such as Sockets, provided by the operating system. These messaging protocols used for inter-process communication are often vulnerable to attack; thus, when writing an application, we must always assume that the process at the other end of our communication channel could be hostile.

1.4.5 Insecure File Operations

In addition to time-of-check–time-of-use problems, many other file operations are insecure. Programmers often make assumptions about the ownership, location, or attributes of a file that might not be true. For example, we might assume that we can always write to a file created by our program. However, if an attacker can change the permissions or flags on that file after we create it, and if we fail to check the result code after a write operation, we will not detect the fact that the file has been tampered with.

Examples of insecure file operations include:

- *writing to or reading from a file in a location writable by another user*
- *failing to make the right checks for file type, device ID, links, and other settings before using a file*

- *failing to check the result code after a file operation*
- *assuming that if a file has a local pathname, it has to be a local file*

1.4.6 Access Control Problems

Access control is the process of controlling who is allowed to do what. This ranges from controlling physical access to a computer—keeping our servers in a locked room, for example—to specifying who has access to a resource (a file, for example) and what they are allowed to do with that resource (such as read only). Some access control mechanisms are enforced by the operating system, some by the individual application or server, some by a service (such as a networking protocol) in use. Many security vulnerabilities are created by the careless or improper use of access controls, or by the failure to use them at all.

Much of the discussion of security vulnerabilities in the software security literature is in terms of privileges, and many exploits involve an attacker somehow gaining more privileges than they should have. Privileges, also called permissions, are access rights granted by the operating system, controlling who is allowed to read and write files, directories, and attributes of files and directories (such as the permissions for a file), who can execute a program, and who can perform other restricted operations such as accessing hardware devices and making changes to the network configuration.

Of particular interest to attackers is the gaining of root privileges, which refers to having the unrestricted permission to perform any operation on the system. An application running with root privileges can access everything and change anything. Many security vulnerabilities involve programming errors that allow an attacker to obtain root

privileges. Some such exploits involve taking advantage of buffer overflows or race conditions, which in some special circumstances allow an attacker to escalate their privileges. Others involve having access to system files that should be restricted or finding a weakness in a program—such as an application installer—that is already running with root privileges. For this reason, it's important to always run programs with as few privileges as possible. Similarly, when it is necessary to run a program with elevated privileges, we should do so for as short a time as possible [2].

Much access control is enforced by applications, which can require a user to authenticate before granting authorization to perform an operation. Authentication can involve requesting a user name and password, the use of a smart card, a biometric scan, or some other method. If an application calls the OS X Authorization Services application interface to authenticate a user, it can automatically take advantage of whichever authentication method is available on the user's system. Writing our own authentication code is a less secure alternative, as it might afford an attacker the opportunity to take advantage of bugs in our code to bypass our authentication mechanism, or it might offer a less secure authentication method than the standard one used on the system.

Digital certificates are commonly used—especially over the Internet and with email—to authenticate users and servers, to encrypt communications, and to digitally sign data to ensure that it has not been corrupted and was truly created by the entity that the user believes to have created it. Incorrect or careless use of digital certificates can lead to security vulnerabilities. For example, a server administration program

shipped with a standard self-signed certificate, with the intention that the system administrator would replace it with a unique certificate. However, many system administrators failed to take this step, with the result that an attacker could decrypt communication with the server. [CVE-2004-0927]

It's worth noting that nearly all access controls can be overcome by an attacker who has physical access to a machine and plenty of time. For example, no matter what you set a file's permissions to, the operating system cannot prevent someone from bypassing the operating system and reading the data directly off the disk. Only restricting access to the machine itself and the use of robust encryption techniques can protect data from being read or corrupted under all circumstances.

1.4.7 Secure Storage and Encryption

Encryption can be used to protect a user's secrets from others, either during data transmission or when the data is stored. OS X provides a variety of encryption-based security options, such as

- *FileVault*
- *the ability to create encrypted disk images*
- *keychain*
- *certificate-based digital signatures*
- *encryption of email*
- *SSL/TLS secure network communication*
- *Kerberos authentication*

The list of security options in iOS includes

- *passcode to prevent unauthorized use of the device*
- *data encryption*

- *the ability to add a digital signature to a block of data*
- *keychain*
- *SSL/TLS secure network communication*

Each service has appropriate uses, and each has limitations. For example, FileVault, which encrypts the contents of a user's root volume (in OS X v10.7 and later) or home directory (in earlier versions), is a very important security feature for shared computers or computers to which attackers might gain physical access, such as laptops. However, it is not very helpful for computers that are physically secure but that might be attacked over the network while in use, because in that case the home directory is in an unencrypted state and the threat is from insecure networks or shared files. Also, FileVault is only as secure as the password chosen by the user—if the user selects an easily guessed password, or writes it down in an easily found location, the encryption is useless.

It is a serious mistake to try to create our own encryption method or to implement a published encryption algorithm ourselves unless we are already an expert in the field. It is extremely difficult to write secure, robust encryption code that generates unbreakable cipher-text, and it is almost always a security vulnerability to try. For OS X, if we need cryptographic services beyond those provided by the OS X user interface and high-level programming interfaces, we can use the open-source CSSM Cryptographic Services Manager. For iOS, the development APIs should provide all the services we need.

1.4.8 Social Engineering

Often the weakest link in the chain of security features protecting a user's data and software is the user himself. As developers eliminate buffer overflows, race conditions, and other security vulnerabilities, attackers increasingly concentrate on fooling users into executing malicious code or handing over passwords, credit-card numbers, and other private information. Tricking a user into giving up secrets or into giving access to a computer to an attacker is known as social engineering.

For example, in February of 2005, a large firm that maintains credit information, Social Security numbers, and other personal information on virtually all U.S. citizens revealed that they had divulged information on at least 150,000 people to scam artists who had posed as legitimate businessmen. According to Gartner (http://www.gartner.com), phishing attacks cost U.S. banks and credit card companies about $1.2 billion in 2003, and this number is increasing. They estimate that between May 2004 and May 2005, approximately 1.2 million computer users in the United States suffered losses caused by phishing [2].

Software developers can counter such attacks in two ways: through educating their users and through clear and well-designed user interfaces that give users the information they need to make informed decisions [1].

CHAPTER 2

LITERATURE SURVEY

2.1 Introduction to Survey Report

In April 7, 2014 The Heartbleed Bug was independently discovered by a team of security engineers (Riku, Antti, and Matti, 2014) at Codenomicon and Neel Mehta of Google Security, who first reported it to the OpenSSL team. The security engineers did not have an idea of the vulnerability until the team found heartbleed bug while improving the Safeguard features. This was the city Codenomicon's Defense security testing tools and reported this bug to the NCSC-FI for vulnerability coordination and reporting to OpenSSL team [5].

In addition, Bloomberg (2014) accused the U.S National Security Agency (NSA) of knowing the Heartbleed Bug for the last two years. Although, the report says the NSA was using it to gain information instead of disclosing it to the OpenSSL developer. After the NSA declining to comment to report of knowing about the Heartbleed Bug, NSA also denied that they were aware of Heartbleed Bug until the vulnerability was made public by the private security engineering of Google. Overall, the questions remain about whether anyone from the NSA or U.S government might have exploited the code for their benefits before published to the public.

The Heartbleed Bug is not a virus, it's not a worm or a malicious code, and it has nothing to do with the Man-in-the-Middle, but it's a simple programming mistake. However, the Heartbleed Bug is a serious vulnerability in the most popular OpenSSL cryptographic software library. This software allows anyone with little knowledge to steal the information such as the names and passwords of the users and the actual content protected, under normal conditions, by the SSL/TLS encryption used to secure the internet. In addition, the code of the Heartbleed Bug is available to the public and there are several sites that have tutorials to teach the use of the software, therefore this vulnerability is most critical.

The purposes of the SSL/TLS are to provide communication security and privacy over the internet for applications such as web, email, VPNs and social media [5]. Smartphones are the best practical example of client side attack, which lead to Blackberry (Z10) products to be vulnerable to Heartbleed Bug, in contrast of Apple's iOS devices are not affected by OpenSSL. There are other devices affected by Heartbleed such as; IP Phones, Routers, Medical Devices and Smart TV sets. In addition, about 34 percent of Android devices run on version 4.1.x of the mobile OS, which according to Google millions of Android smartphones never, or only rarely receive available updates that patch dangerous security defects. For that reason, Android users should download Heartbleed Detector, a free application developed by Lookout.

The Heartbleed Bug attack works in several steps: First, the attacker creates a custom Heartbleed. Second, the packet is transmitted to vulnerable OpenSSL web server. Third web server processes packet. Fourth, the code grabs up 64KB of extra memory and hopes of capturing

something sensitive from memory. Fifth, web server responds by sending a packet back which knowingly includes this extra sensitive data. Sixth, attacker analyzes packets to see if there is anything interesting, if not reruns attack to capture more memory. Lastly, if web server's certificates private key is captured, it can be used to decrypt current and historical user data and credentials. Overall, is not complex to use the Heartbleed software. As mentioned before, any Heartbleed based attacks are not traceable, due that the problem has existed for the past 2 years without the knowledge of the public. Most server operators use a vulnerable method of the OpenSSL versions 1.0.1 – 1.0.1f and likely don't have enough logs/monitoring to determine whether a site was compromised.

The Heartbleed bug reflects one of the most catastrophic vulnerabilities during the OpenSSL history for several reasons: it allowed attackers to retrieve private information and user data, it was easy to exploit and HTTPS and other TLS services have become increasingly popular by the resulting in more affected services [6].In addition, Stephen Solis-Reyes 19 year-old from Canada was arrested for exploiting the Heartbleed Bug to attack the website of the Canada Revenue Agency. As result, of the attack, Mr. Solis-Reyes had stolen 900 social insurances numbers (Elsevier, 2014). According, to Ivan Ristic, director of engineering at Qualys, has claimed that the percentage of websites vulnerable to the flaw had dropped from 25 percent since the bug was discovered. "Assistant Research Scientist Dave Levin and Assistant Professor of Electrical and Computer Engineering Tudor Dumitras were part of a team that analyzed the most popular websites in the United States-more than one million sites were examined-to better understand

the extent to which systems administrators followed specific protocols to fix the problem"(NewsRx, 2014) [5].

2.2 General Survey

Who and what caused Heartbleed Bug? This question is answered with two graphics, displaying the bad code and the good code. The programmer Robin Seggelmann, a 31 year old based in Germany, submitted the code. The purpose of the software was to enable a function called "Heartbeat" in OpenSSL. This software package was to be used by nearly half of all web servers to enforce the connections. "In one of the new features, unfortunately, I missed validating a variable containing a length" (Seggelmann, 2012). In addition, the code went undetected by several code reviewers and everyone else for over two years. The graphics below shows the c- language code for the Heartbeat message in the OpenSSL source code. In the first graphic, it shows the data structure and the length of the message is given as payload_length.

As it shows below in the Graphic 1(Figure 0), the incoming data contains a payload length "payload", the mistake of the code is that it trusts the request without bounds checks. OpenSSL then allocates a buffer for its response, and copies "payload" data bytes from the pointer "pl" into it. As result, there's no "if statement" to make sure that there are actually "payload" bytes in data, or that this is in bounds. Since, there is no "if statement" the attacker gets a 64KB of data in length from main memory. When the attacker gets the 64KB of data the connection is no longer secure between servers and computers [7].

As stated by (Balon, 2014) "Think of the server's memory like your mind," "Whatever is going through your mind right now, what you see in front of you, my words to you, whatever else you're thinking about -- maybe a hotdog pops in there -- all that information can be scraped off the server, through the Heartbleed bug," (Balon, 2014).

On the other hand, Graphic 2(Figure 0) shows the correct code with the "if statement" placed in the correct place. However, by making the correction of the code it does not guarantee that our server is secure and is no longer vulnerable. In order, to have a secure server or routers the security technician must take the following actions; upgrade your server to the latest version of OpenSSL, reissue and then revoke all certificates used with the vulnerable version of OpenSSL, and upgrade your security patches. As social media and online shopping user such as; Facebook, Google, eBay, Instagram and other sites that require user credentials may have to change our password if we haven't change within the past 6 months.

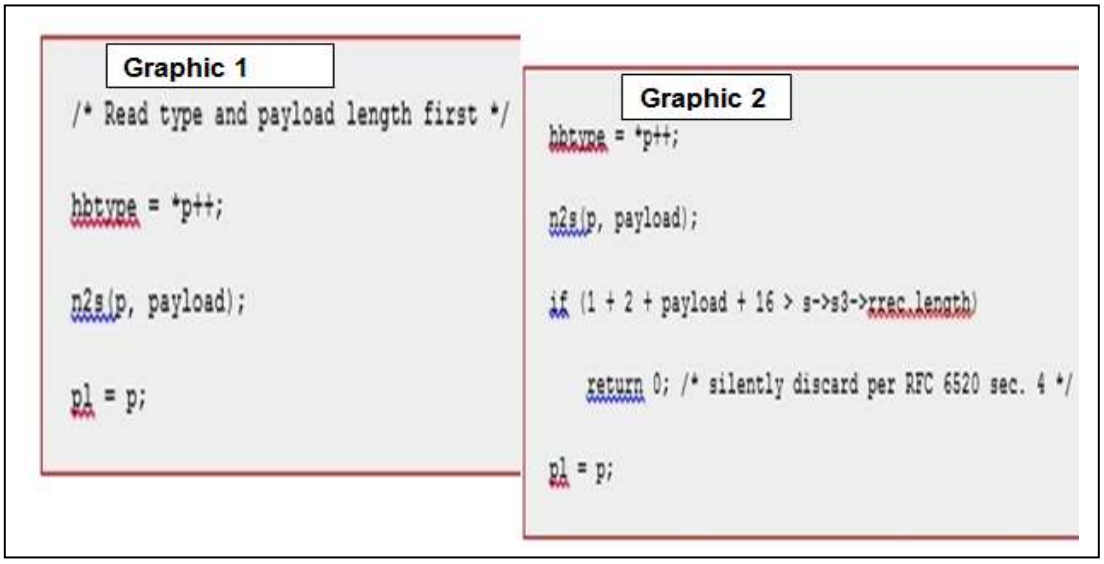


Figure 0: Graphic 1 and 2 shows the Heartbleed code

CHAPTER 3

THEORETICAL EXPLAINATIONS

3.1 How The Heartbeat Works

The heartbeat extension protocol consists of two message types: HeartbeatRequest message and HeartbeatResponse message and the extension protocol depends on which TLS protocol is being used as describe below:

3.1.1 When Using Reliable Transport Protocol

One side of the peer connection sends a HeartbeatRequest message to the other side. The other side of the connection should immediately send a HeartbeatResponse message. This makes one successful Heartbeat and thus, keeping connection alive – this is called 'keep-alive' functionality. If no response is received within a specified timeout, the TLS connection is terminated.

3.1.2 Unreliable Transport Protocol

One side of the peer connection sends HeartbeatRequest message to the other side. The other side of the connection should immediately send a HeartbeatResponse message. If no response is received within specified timeout another HeartbeatRequest message is retransmitted. If expected response is not received for specified number of retransmissions, the DTLS (Datagram Transport Layer Security) connection is terminated.

When a receiver receives a HeartbeatRequest message, the receiver should send back an exact copy of the received message in the HeartbeatResponse message. The sender verifies that the HeartbeatResponse message is same as what was originally sent. If it is same, the connection is kept alive. If the response does not contain the same message, the HeartbeatRequest message is retransmitted for a specified number of retransmissions [7].

3.2 Data Leakage Leading to Heartbleed

There is a bug in the implementation of the Heartbeat reply to the received Heartbeat request message. Heartbeat reply copies the received payload to the Heartbeat response message to verify that the secured connection is still active, without checking if the payload length is same as the length of the request payload data.

The problem here is that the OpenSSL heartbeat response code does not check to make sure that the payload length field specified in the heartbeat request message matches the actual length of the payload.

If the heartbeat request payload length field is set to a value larger than the actual payload, it would result in a return of the payload followed by whatever contents are currently contained in active memory buffer, beyond the end of the payload. A heartbeat request the payload length can be set to a maximum value of 65535 bytes. Therefore the bug in the OpenSSL heartbeat response code could copy as much as 65535 bytes from the machine's memory and send it to the requestor [6].

This bug is illustrated below in “Figure 1: Memory Leak.

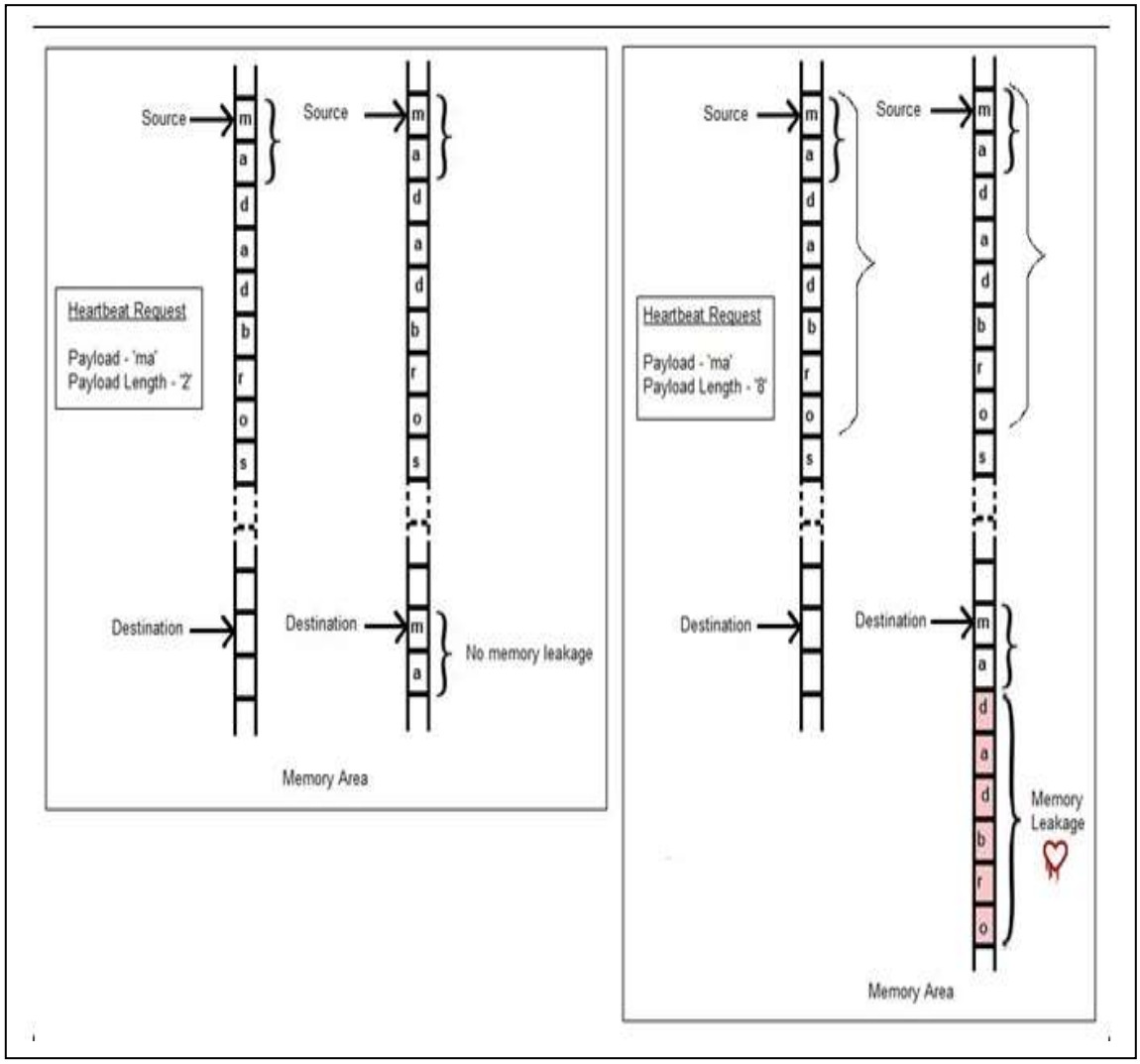


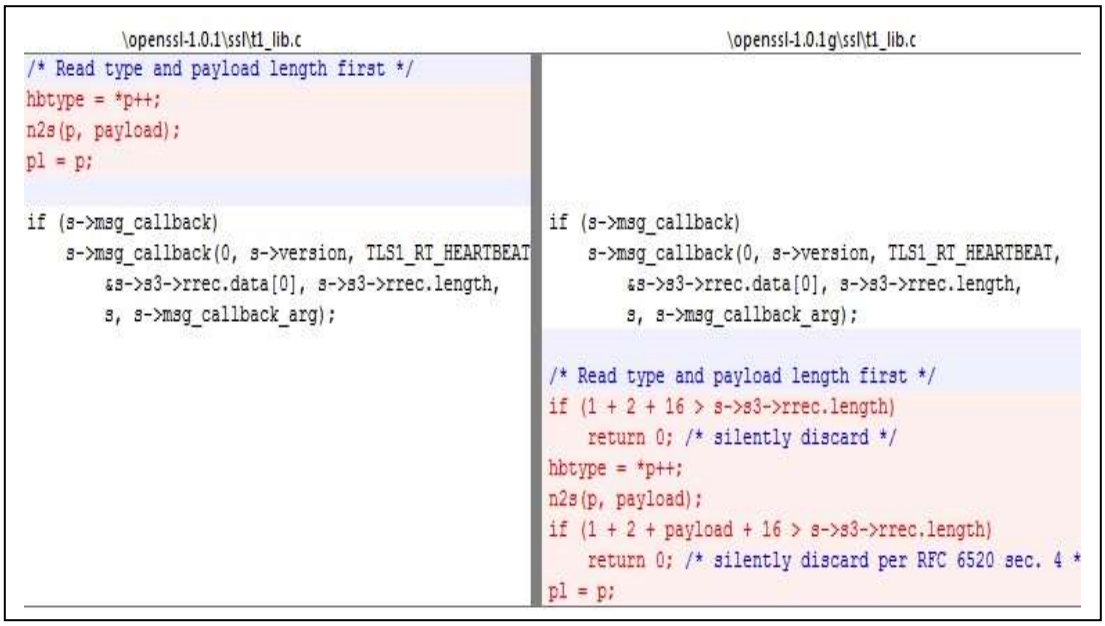
Figure 1: Memory Leak

"Figure 1: Memory leak" shows that when the request payload data is 'ma' and payload length is '2' then 2 bytes from source (i.e. 'ma') is copied to the 'destination' memory area. But when the request payload data is 'ma' and payload length falsely indicates that it is 8 bytes instead of 2, 8 bytes (i.e. 'madadbro') from the 'source' memory area to the 'destination' memory area. This 'destination' data is finally sent to the

requestor, causing the memory leak that is now known as the Heartbleed bug [9].

3.3 Code Fix

"Figure 2: The OpenSSL code fix for the Heartbleed bug" shows the change in OpenSSL's file t1_lib.c between version 1.0.1 and OpenSSL version 1.0.1g that was made to fix the Heartbleed bug [7].



\openssl-1.0.1\ssl\t1_lib.c

```
/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;

if (s->msg_callback)
    s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT
        &s->s3->rrec.data[0], s->s3->rrec.length,
        s, s->msg_callback_arg);
```

\openssl-1.0.1g\ssl\t1_lib.c

```
if (s->msg_callback)
    s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
        &s->s3->rrec.data[0], s->s3->rrec.length,
        s, s->msg_callback_arg);

/* Read type and payload length first */
if (1 + 2 + 16 > s->s3->rrec.length)
    return 0; /* silently discard */
hbtype = *p++;
n2s(p, payload);
if (1 + 2 + payload + 16 > s->s3->rrec.length)
    return 0; /* silently discard per RFC 6520 sec. 4 *
pl = p;
```

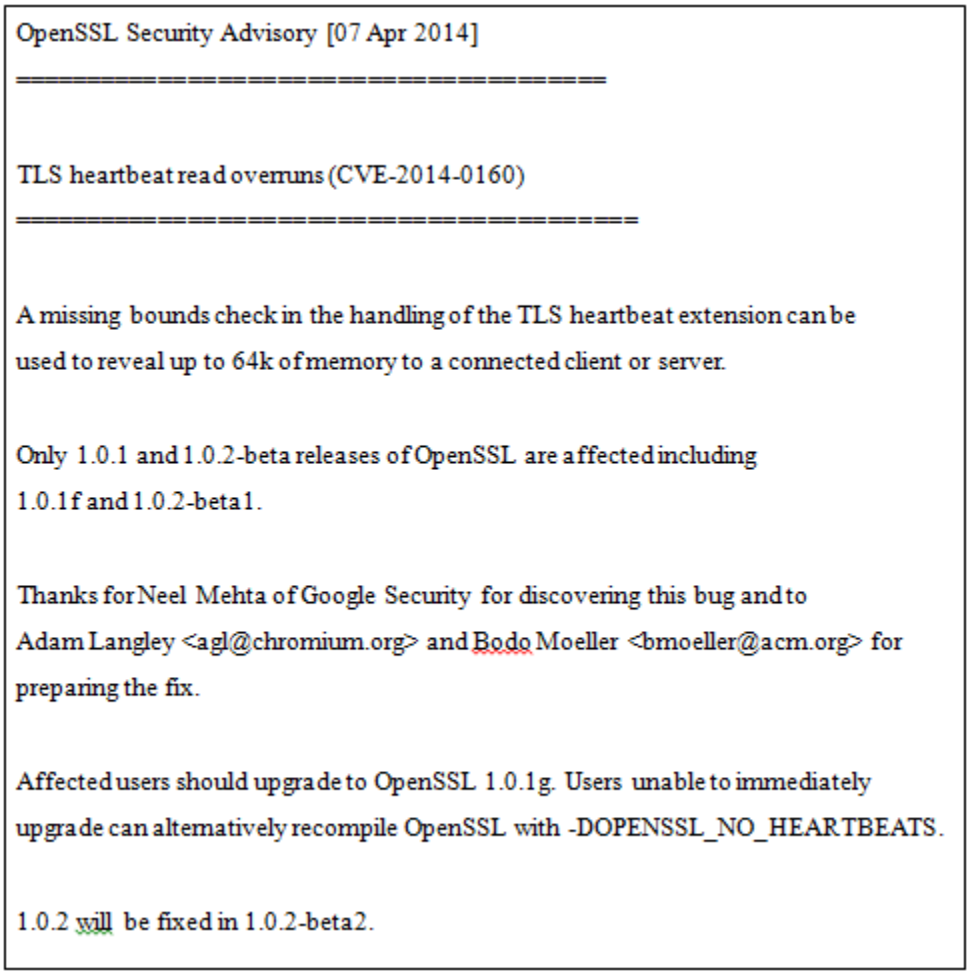
Figure 2: The OpenSSL code fix for the Heartbleed bug

This code fix has two tasks to perform:

First, it checks to determine if the length of the payload is zero or not. It simply discards the message if the payload length is 0.

The second task performed by the bug fix makes sure that the heartbeat payload length field value matches the actual length of the request payload data. If not, it discards the message.

The official notice about the bug was published by the OpenSSL group at https://www.openssl.org/news/secadv_20140407.txt and is reproduced in "Figure 3: OpenSSL Security Advisory."



OpenSSL Security Advisory [07 Apr 2014]

==========

TLS heartbeat read overruns (CVE-2014-0160)

==========

A missing bounds check in the handling of the TLS heartbeat extension can be used to reveal up to 64k of memory to a connected client or server.

Only 1.0.1 and 1.0.2-beta releases of OpenSSL are affected including 1.0.1f and 1.0.2-beta1.

Thanks for Neel Mehta of Google Security for discovering this bug and to Adam Langley <agl@chromium.org> and Bodo Moeller <bmoeller@acm.org> for preparing the fix.

Affected users should upgrade to OpenSSL 1.0.1g. Users unable to immediately upgrade can alternatively recompile OpenSSL with -DOPENSSL_NO_HEARTBEATS.

1.0.2 will be fixed in 1.0.2-beta2.

Figure 3: OpenSSL Security Advisory

3.4 Real-World Impact of Heartbleed

By exploiting the Heartbleed vulnerability, an attacker can send a Heartbeat request message and retrieve up to 64 KB of memory from the victim's server. The contents of the retrieved memory depends on what's in memory in the server at the time, but could potentially contain usernames, passwords, session IDs or secret private keys or other

sensitive information. Following figure illustrates how an attacker can exploit this vulnerability. This attack can be made multiple times without leaving any trace of it. "Figure 4: Exploiting the Heartbleed vulnerability" illustrates how an attacker can exploit the Heartbleed vulnerability.

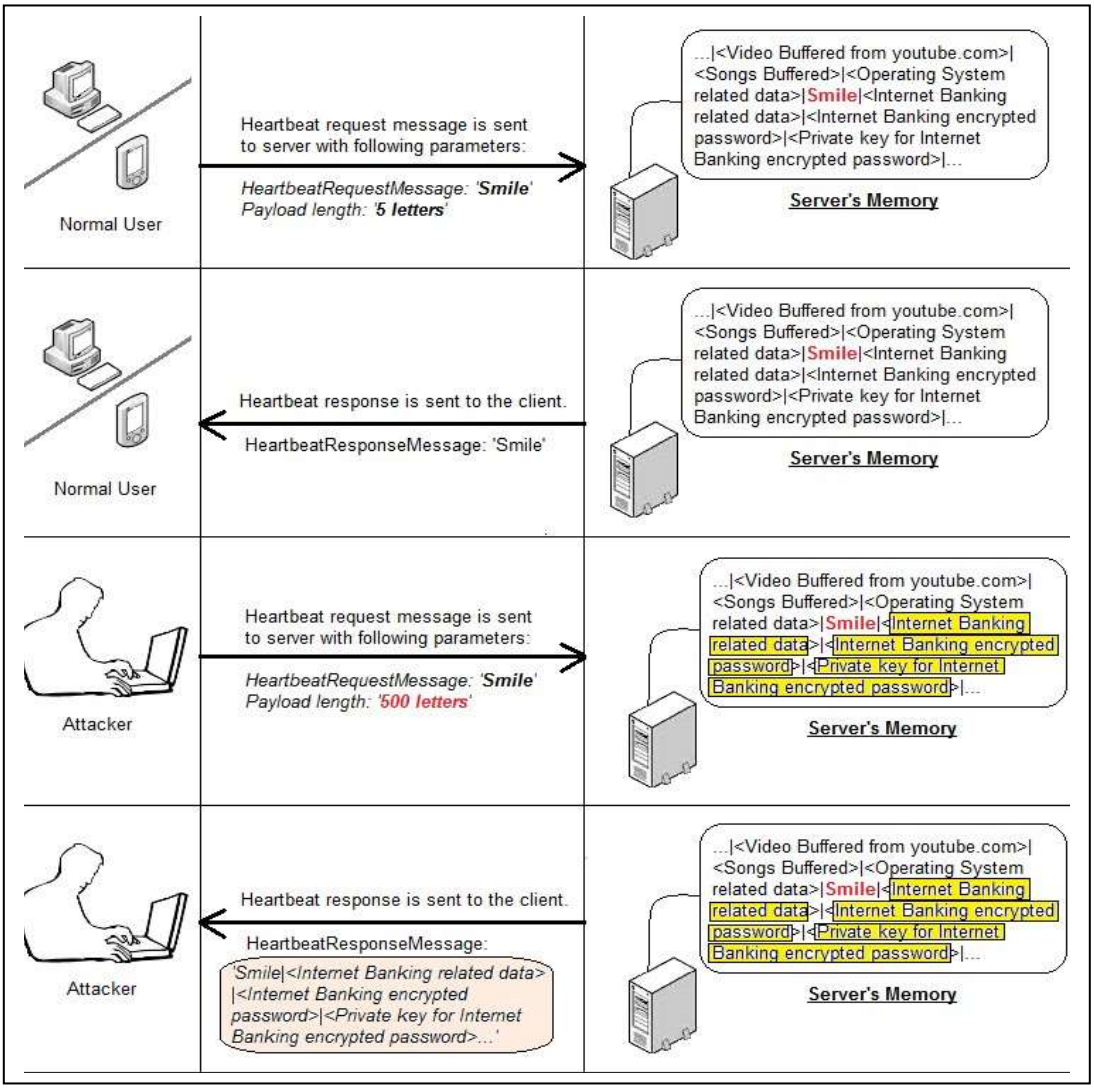


Figure 4: Exploiting the Heartbleed vulnerability

3.5 Factors to Determine Severity of a Vulnerability- Common Vulnerability Scoring System(CVSS)

The data used in this section comes from many different sources. The main reference source is the National Vulnerability Database (NVD), which includes Information for all Common Vulnerabilities and Exposures (CVEs). As of May 2015, there are close to 69,000 CVEs in the database. Connected to each CVE is also a list of external references to exploits, bug trackers, vendor pages, etc. Each CVE comes with some Common Vulnerability Scoring System (CVSS) metrics and parameters, which can be found in Table 1. A CVE- number is of the format CVE-Y-N, with a four number year Y, and a 4-6 number identifier N per year. Major vendors are pre-assigned ranges of CVE-numbers to be registered in the oncoming year, which means that CVE-numbers are not guaranteed to be used or to be registered in consecutive order [9].

The data from the NVD only includes the base CVSS Metric parameters seen in Table 1. In the official guide for CVSS metrics, Mell et al. (2007) describes that there are similar categories for Temporal Metrics and Environmental Metrics. The first one includes categories for Exploitability, Remediation Level, and Report Confidence. The latter includes Collateral Damage Potential, target Distribution, and Security Requirements. Part of this data is sensitive and cannot be publicly disclosed.

TABLE 1: CVSS (version 2) Base Metrics, with definitions from Mell et al. (2007)

Parameter	Values	Description
CVSS Score	0-10 [Low (0.1 -3.9), Medium (4.0 – 6.9), High (7.0 – 8.9), Critical (9.0 – 10.0)]	This value is calculated based on the next six parameters, with a formula (Mell et al., 2007).
Access Vector	Local Adjacent Network	The access vector (AV) determines how vulnerability can be exploited. A local attack requires physical access to the computer or a shell account. Vulnerability with Network access is also called remotely exploitable.
Access Complexity	Low Medium High	The access complexity (AC) classifies the difficulty to exploit the vulnerability.
Authentication	None Single Multiple	The authentication (Au) categorizes the number of times that an attacker must authenticate to a target to exploit it, but does not measure the difficulty of the authentication process itself.
Confidentiality	None Partial Complete	The confidentiality (C) metric assorts the impact of the confidentiality, and amount of information access and disclosure. This may include partial or full access to file systems and/or database tables.
Integrity	None Partial Complete	The integrity (I) metric categorizes the impact on the integrity of the exploited system. For example, if the remote attack is able to partially or fully modify information in the exploited system.
Availability	None Partial Complete	The availability (A) metric categorizes the impact on the availability of the target system. Attacks that consume network bandwidth, processor cycles, memory or any other resources affect the availability of a system.

3.6 Naive Bayes Classification

Naive Bayes is a classification algorithm for binary (two-class) and multi-class classification problems. The technique is easiest to understand when described using binary or categorical input values [8].

It is called naive Bayes or idiot Bayes because the calculation of the probabilities for each hypothesis is simplified to make their calculation tractable. Rather than attempting to calculate the values of each attribute value P (d1, d2, d3|h), they are assumed to be conditionally independent given the target value and calculated as P (d1|h) * P (d2|H) and so on.

This is a very strong assumption that is most unlikely in real data, i.e. that the attributes do not interact. Nevertheless, the approach performs surprisingly well on data where this assumption does not hold.

3.6.1 Representation Used By Naive Bayes Models

The representation for naive Bayes is probabilities.

A list of probabilities is stored to file for a learned naive Bayes model. This includes:

- *Class Probabilities*—the probabilities of each class in the training dataset.
- *Conditional Probabilities*—the conditional probabilities of each input value given each class value.

3.6.2 Learn a Naive Bayes Model from Data

Learning a naive Bayes model from training data is fast.

Training is fast because only the probability of each class and the probability of each class given different input (x) values need to be calculated. No coefficients need to be fitted by optimization procedures.

3.6.3 Calculating Class Probabilities

The class probabilities are simply the frequency of instances that belong to each class divided by the total number of instances.

For example in a binary classification the probability of an instance belonging to class 1 would be calculated as:

```
P(class=1) = count(class=1) / (count(class=0) + count(class=1))
```

In the simplest case each class would have the probability of 0.5 or 50% for a binary classification problem with the same number of instances in each class.

3.6.4 Calculating Conditional Probabilities

The conditional probabilities are the frequency of each attribute value for a given class value divided by the frequency of instances with that class value.

For example, if a "weather" attribute had the values "sunny" and "rainy" and the class attribute had the class values "go-out" and "stay-home", then the conditional probabilities of each weather value for each class value could be calculated as:

- P(weather=sunny|class=go-out) = count(instances with weather=sunny and class=go-out) / count(instances with class=go-out)
- P(weather=sunny|class=stay-home) = count(instances with weather=sunny and class=stay-home) / count(instances with class=stay-home)
- P(weather=rainy|class=go-out) = count(instances with weather=rainy and class=go-out) / count(instances with class=go-out)
- P(weather=rainy|class=stay-home) = count(instances with weather=rainy and class=stay-home) / count(instances with class=stay-home)

3.6.5 Make Predictions with a Naive Bayes Model

Given a naive Bayes model, you can make predictions for new data using Bayes theorem.

$$MAP(h) = max(P(d|h) * P(h))$$

Using our example above, if we had a new instance with the weather of sunny, we can calculate:

go-out = P(weather=sunny|class=go-out) * P(class=go-out)
stay-home = P(weather=sunny|class=stay-home) * P(class=stay-home)

We can choose the class that has the largest calculated value. We can turn these values into probabilities by normalizing them as follows:

P(go-out|weather=sunny) = go-out / (go-out + stay-home)
P(stay-home|weather=sunny) = stay-home / (go-out + stay-home)

If we had more input variables we could extend the above example. For example, pretend we have a "car" attribute with the values "working" and "broken". We can multiply this probability into the equation.

For example below is the calculation for the "go-out" class label with the addition of the car input variable set to "working":

go-out = P(weather=sunny|class=go-out) * P(car=working|class=go-out) * P(class=go-out)

3.6.6 Gaussian Naive Bayes

Naive Bayes can be extended to real-valued attributes, most commonly by assuming a Gaussian distribution.

This extension of naive Bayes is called Gaussian Naive Bayes. Other functions can be used to estimate the distribution of the data, but the Gaussian (or Normal distribution) is the easiest to work with because you only need to estimate the mean and the standard deviation from your training data.

3.6.7 Representation for Gaussian Naive Bayes

Above, we calculated the probabilities for input values for each class using a frequency. With real-valued inputs, we can calculate the

mean and standard deviation of input values (x) for each class to summarize the distribution.

This means that in addition to the probabilities for each class, we must also store the mean and standard deviations for each input variable for each class.

3.6.8 Learn a Gaussian Naive Bayes Model from Data

This is as simple as calculating the mean and standard deviation values of each input variable (x) for each class value.

$$mean(x) = 1/n * sum(x)$$

Where n is the number of instances and x are the values for an input variable in your training data.

We can calculate the standard deviation (sd) using the following equation:

$$standard\ deviation(x) = sqrt(1/n * sum(xi-mean(x)^2))$$

This is the square root of the average squared difference of each value of x from the mean value of x, where n is the number of instances, sqrt( ) is the square root function, sum( ) is the sum function, xi is a specific value of the x variable for the i'th instance and mean(x) is described above, and ^2 is the square.

3.6.9 Make Predictions with a Gaussian Naive Bayes Model

Probabilities of new x values are calculated using the Gaussian Probability Density Function (PDF).

When making predictions these parameters can be plugged into the Gaussian PDF with a new input for the variable, and in return the Gaussian PDF will provide an estimate of the probability of that new input value for that class.

```
pdf(x, mean, sd) = (1 / (sqrt(2 * PI) * sd)) * exp(-((x-mean^2)/(2*sd^2)))
```

Where pdf(x) is the Gaussian PDF, sqrt( ) is the square root, mean and sd are the mean and standard deviation calculated above, PI is the numerical constant, exp( ) is the numerical constant e or Euler's number raised to power and x is the input value for the input variable.

We can then plug in the probabilities into the equation above to make predictions with real-valued inputs.

For example, adapting one of the above calculations with numerical values for weather and car:

```
go-out = P(pdf(weather)|class=go-out) * P(pdf(car)|class=go-out) * P(class=go-out)
```

CHAPTER 4

PROPOSED WORK

4.1 Algorithm for Predicting Severity/Threat Of Exploitation Using Naive Bayes Approach

- Convert the data set into a frequency table.
- Create Likelihood table by finding the probabilities, like probability of High threat of exploitation is (4/7) = 0.57 and probability of Low threat of exploitation is (3/7) = 0.43.
- Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

4.2 Frequency Table for Some Common Vulnerabilities Based on CVSS (Version2) Parameters

The values for CVSS (Version2) parameters: CVSS Score, Access Vector, Access, Complexity, Authentication, Confidentiality, Integrity and Availability for some common types of vulnerabilities such as PhpMyAdmin Reflected cross- Site Scripting Vulnerability (CVE-2013-1937), MySQL Stored SQL Injection (CVE-2013-0375), SSL v3 POODLE Vulnerability (CVE_2014-3568), VMWare Guest to Host Escape Vulnerability (CVE-2012-1516), Apache Tomcat XML Parser Vulnerability (CVE-2009-0783), OpenSSL Heartbleed Vulnerability (CVE-2014-0160) etc. are tabulated in table 2.

TABLE 2: Frequency table for some common vulnerabilities using CVSS (version 2) Base Metrics

Vulnerability	CVSS V2 Base score	Access Vector	Access Complexity	Authentication	Confidentiality	Integrity Impact	Availability Impact	Severity/ Threat of Exploitation
PhpMyAdmin Reflected cross- Site Scripting Vulnerability (CVE-2013-1937)	Medium	Network	Medium	None	None	Partial	None	Low
MySQL Stored SQL Injection (CVE-2013-0375)	Medium	Network	Low	Single	Partial	Partial	None	High
SSL v3 POODLE Vulnerability (CVE_2014-3568)	Medium	Network	Medium	None	Partial	None	None	Low
VMWare Guest to Host Escape Vulnerability (CVE-2012-1516)	Critical	Network	Low	Single	Complete	Complete	Complete	High
Apache Tomcat XML Parser Vulnerability (CVE-2009-0783)	Medium	Local	Low	None	Partial	Partial	Partial	High
Cisco IOS Arbitrary Command Execution Vulnerability (CVE-2012-0384)	High	Network	Medium	Single	Complete	Complete	Complete	High
Apple iWork Denial of Service Vulnerability (CVE-2015-1098)	Medium	Network	Medium	None	Partial	Partial	Partial	Low

OpenSSL Heartbleed Vulnerability (CVE-2014-0160)	Medium	Network	Low	None	Partial	None	None	High
GNU Bourne-Again Shell(Bash) 'ShellShock' Vulnerability (CVE-2014-6271)	Critical	Network	Low	None	Complete	Complete	Complete	High
DNS Kaminsky Bug (CVE-2008-1447)	Medium	Network	Low	None	None	Partial	None	Low
Joomla Directory Traversal Vulnerability (CVE-2010-0467)	Medium	Network	Low	None	Partial	None	None	Low
Cisco Access Control ByPass Vulnerability (CVE-2012-1342)	Medium	Network	Low	None	None	Partial	None	Low
Juniper Proxy ARP Denial of Service Vulnerability (CVE-2013-6014)	Medium	Adjacent	Low	None	None	Complete	None	High
DokuWiki Reflected Cross-Site Scripting Attack (CVE-2014-9253)	Medium	Network	Medium	None	None	Partial	None	Low
Adobe Acrobat Buffer Overflow Vulnerability (CVE-2009-0658)	Critical	Network	Medium	None	Complete	Complete	Complete	High

Microsoft Windows Bluetooth Remote Code Execution Vulnerability (CVE-2011-1265)	High	Network	Low	None	Complete	Complete	Complete	High
Apple ios Security control Bypass vulnerability (CVE-2014-2019)	Medium	Local	Low	None	None	Complete	None	High
SearchBlox Cross-Site Request Forgery Vulnerability (CVE-2015-0970)	Medium	Network	Medium	None	Partial	Partial	Partial	Low
SSL/TLS MITM Vulnerability (CVE-2014-0224)	Medium	Network	Medium	None	Partial	Partial	Partial	Low
Google Chrome ByPass Vulnerability (CVE-2012-5376)	Critical	Network	Low	None	Complete	Complete	Complete	High

4.3 Likelihood Table for Finding the Probabilities(P) Of Various CVSS (version 2) Parameters

The likelihood table for finding the probabilities of various CVSS parameters : CVSS Score, Access Vector, Access, Complexity, Authentication, Confidentiality, Integrity and Availability, as defined in table 1, using the data presented in above frequency table (table 2) is depicted in table 3.

TABLE 3: Likelihood table for calculation of probabilities of CVSS (version 2) parameters

Severity/Threat of Exploitation	
P (High) = 12/21 = 4/7	P (Low) = 9/ 21 = 3/7
CVSS V2 Base score	
P (Low/ High) = 0/12 = 0	P(Low/ Low) = 0/9 = 0
P (Medium/ High) = 6/12 =1/2	P(Medium/ Low) = 9/9 = 1
P (High/ High) = 2/12 = 1/6	P(High/ Low) = 0/9 = 0
P (Critical/ High) = 4/12 = 1/3	P(Critical/ Low) = 0/9 = 0
Access Vector (AV)	
P (Local/ High) = 3/12 = 1/4	P (Local/ Low) = 0/9 = 0
P (Adjacent/ High)= 1/12	P (Adjacent/ Low) = 0/9 = 0
P (network/ High) = 8/12 = 2/3	P (Network/ Low) = 9/9 = 1
Access Complexity (AC)	
P (Low/ High) = 9/12 = 3/4	P (Low/ Low) = 3/9 = 1/3
P (Medium/ High)= 3/12 = 1/4	P (Medium/ Low) = 6/9 = 2/3
P (High/ High) = 0/12 = 0	P (High/ Low) = 0/9 = 0
Authentication (Au)	
P (None/ High) = 9/12 = 3/4	P (None/ Low) = 9/9 = 1
P (Single/ High)= 3/12 = 1/4	P (Single/ Low) = 0/9 = 0
P (Multiple/ High) = 0/12 = 0	P (Multiple/ Low) = 0/9 = 0
Confidentiality Impact (C)	
P (None/ High) = 2/12 = 1/6	P (None/ Low) = 4/9
P (Partial/ High)= 3/12 = 1/4	P (Partial/ Low) = 5/9
P (Complete/ High) = 7/12	P (Complete/ Low) = 0/9 = 0
Integrity Impact (I)	
P (None/ High) = 1/12	P (None/ Low) = 2/9
P (Partial/ High)= 2/12 = 1/6	P (Partial/ Low) = 7/9
P (Complete/ High) = 9/12 = 3/4	P (Complete/ Low) = 0/9 = 0
Availability Impact (A)	
P (None/ High) = 4/12 = 1/3	P (None/ Low) = 6/9 = 2/3
P (Partial/ High)= 1/12	P (Partial/ Low) = 3/9 = 1/3
P (Complete/ High) = 7/12	P (Complete/ Low) = 0/9 = 0

4.4 Using Naive Bayes Equation to Calculate the Posterior Probability for a Sample Class of Vulnerability, to Predict its Severity of Exploitation

Let A be a sample vulnerability with CVSS parameters as:

<Medium, Local, Low, None, Partial, Partial, Partial>

The posterior probability of sample class A, for given set of CVSS parameters, is calculated from table 3 as:

$$P(A/High) \times P(High) = P(Medium/High) \times P(Local/High) \times P(Low/High) \times P(None/High) \times P(Partial/High) \times P(Partial/High) \times P(Partial/High) \times P(High)$$

$$P(A/High) \times P(High) = (1/2) \times (1/4) \times (3/4) \times (3/4) \times (1/4) \times (1/6) \times (1/12) \times (4/7)$$

$$P(A/High) \times P(High) = 36/258048$$

$$P(A/High) \times P(High) = 0.0001395089$$

Now we will calculate P (A/Low) × P (Low) as:

$$P(A/Low) \times P(Low) = P(Medium/Low) \times P(Local/Low) \times P(Low/Low) \times P(None/Low) \times P(Partial/Low) \times P(Partial/Low) \times P(Partial/Low) \times P(Low)$$

$$P(A/Low) \times P(Low) = (9/9) \times (0) \times (1/3) \times (1) \times (5/9) \times (7/9) \times (1/3) \times (3/7)$$

$$P(A/Low) \times P(Low) = 0$$

Now the highest posterior probability is calculated to be:

$$MAX\{P(A/High) \times P(High), P(A/Low) \times P(Low)\} = MAX\{0.0001395089, 0\}$$

$$MAX\{P(A/High) \times P(High), P(A/Low) \times P(Low)\} = 0.0001395089$$

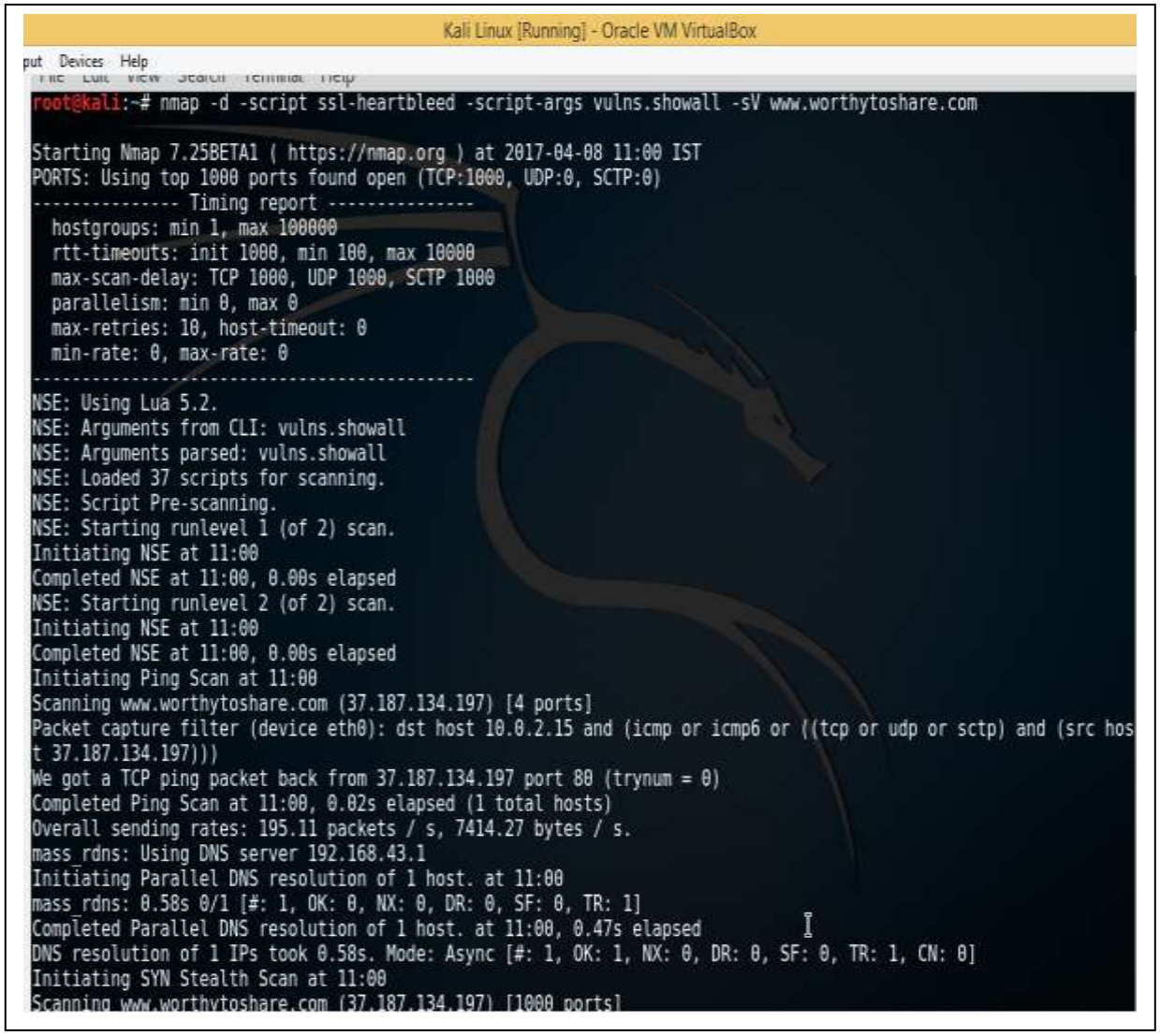
Since {P (A/High) × P (high)} is evaluated to be greater than {P (A/Low) × P (Low)}, hence the sample vulnerability class A with the CVSS parameters as:

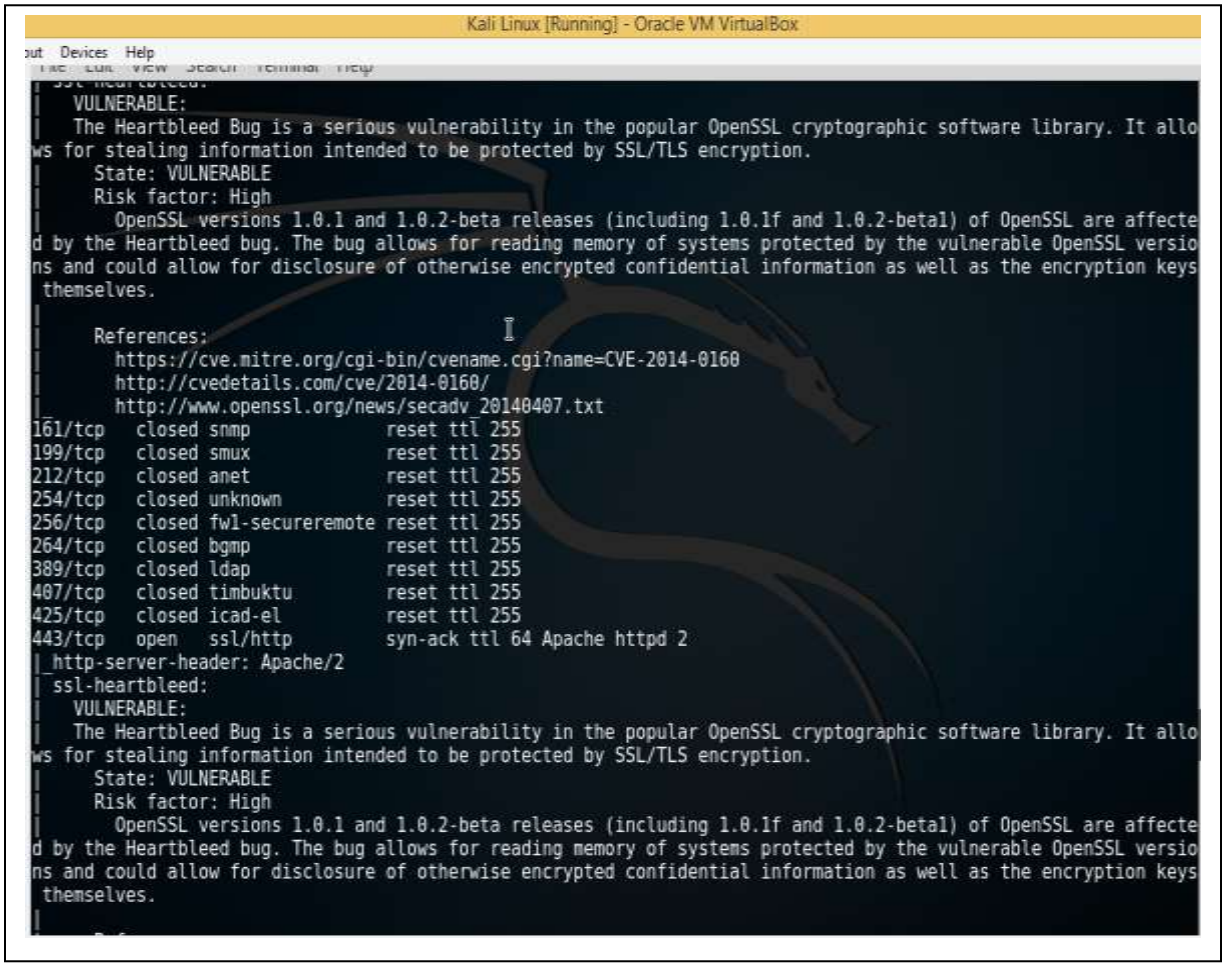
< Medium, Local, Low, None, Partial, Partial, Partial> is predicted to pose high threat of exploitation and thus should quickly be reported for immediate remediation, to prevent the hackers from stealing the valuable data.

CHAPTER 5

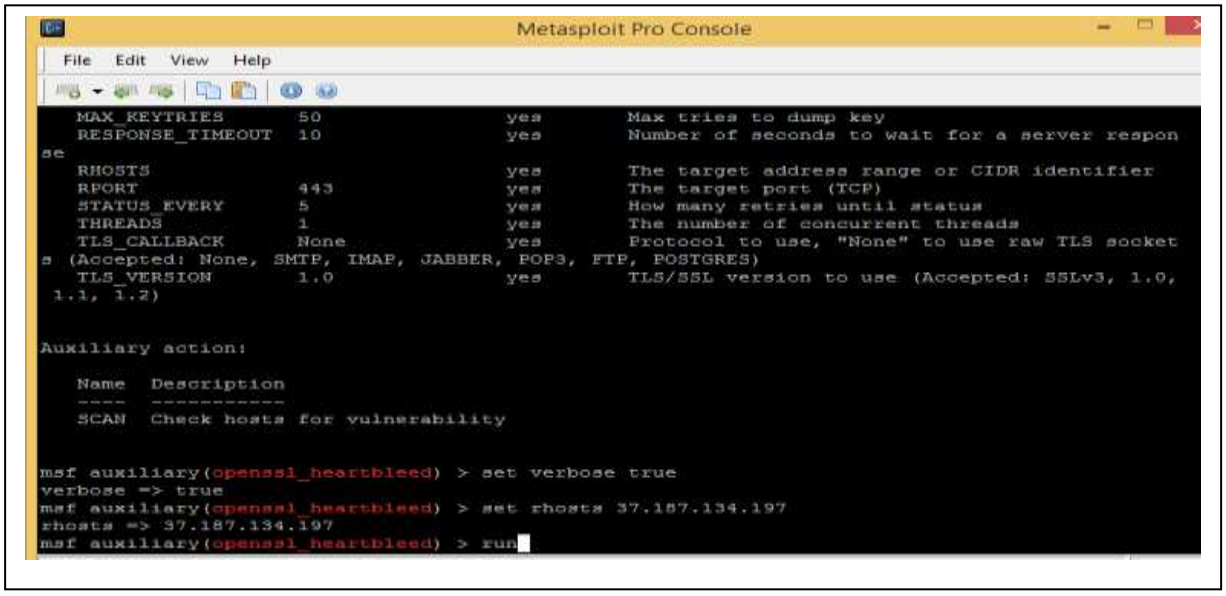
RESULTS AND OUTPUTS

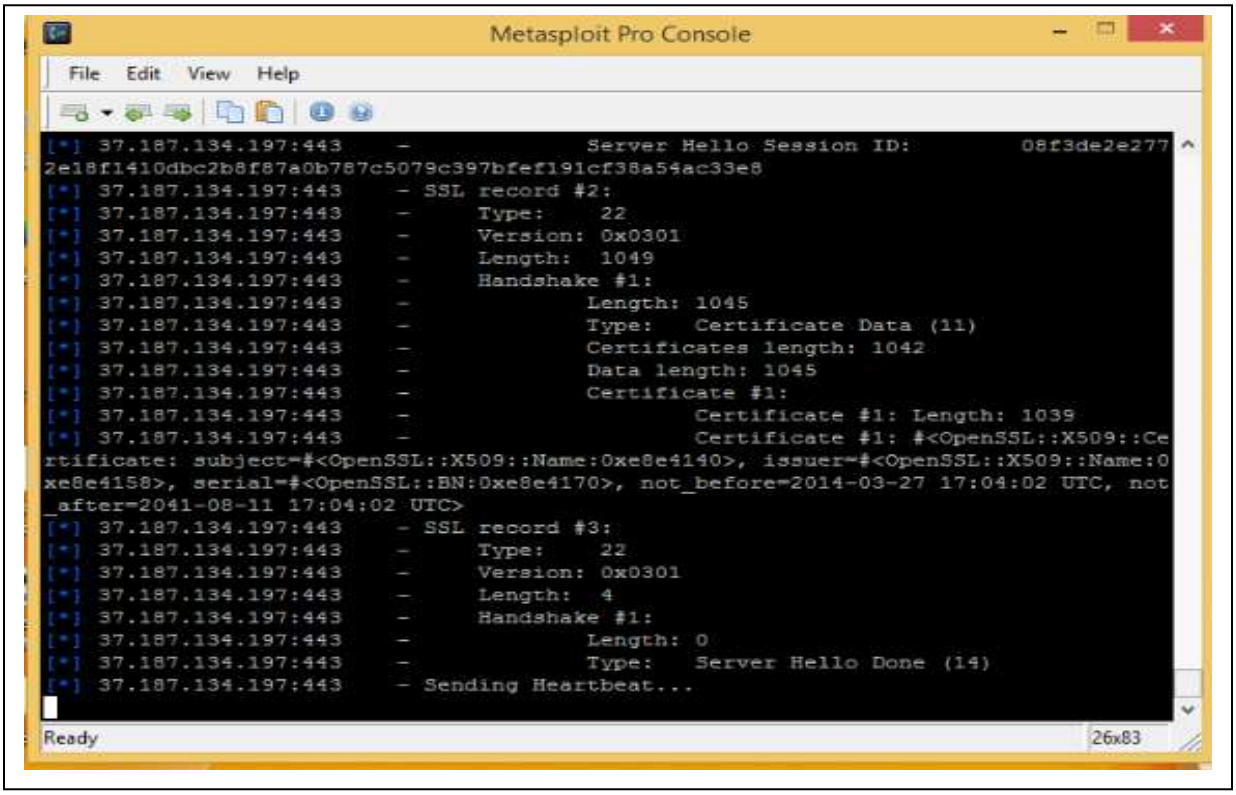
5.1 Checking Heartbleed vulnerability with nmap in Kali Linux

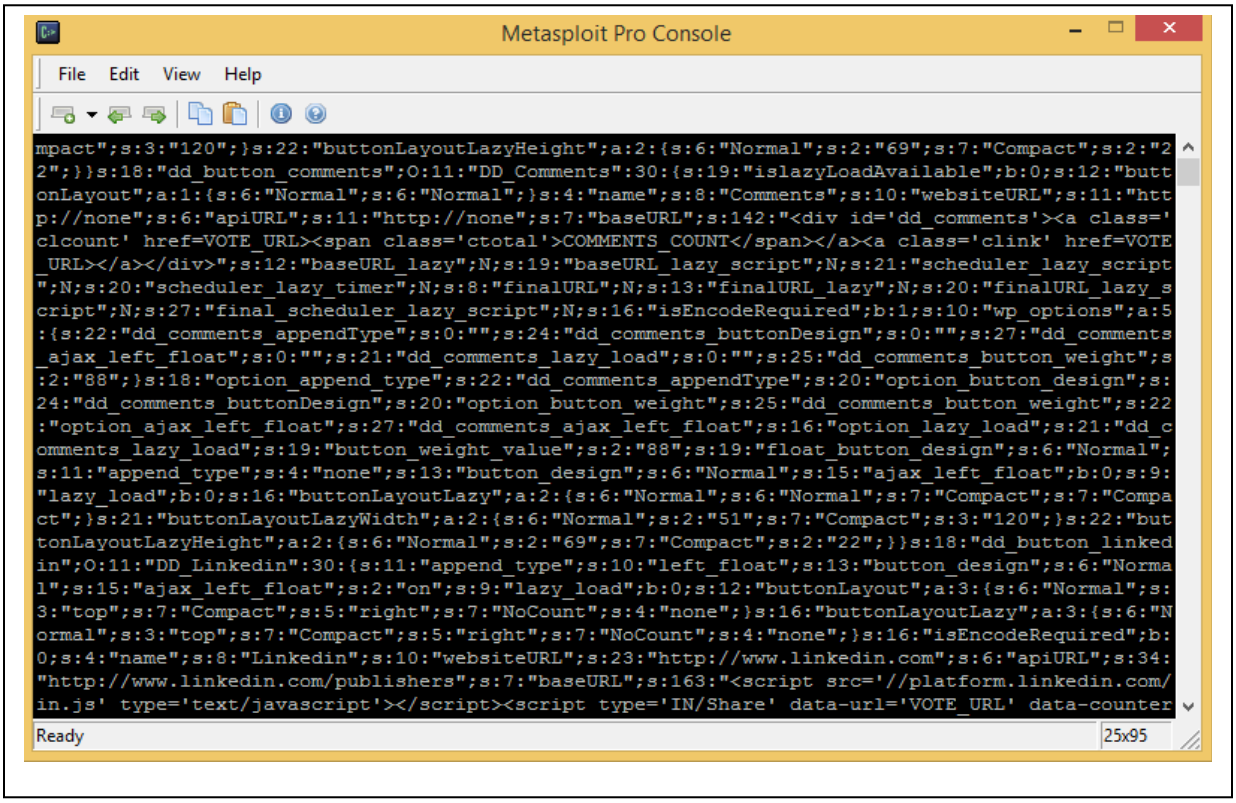




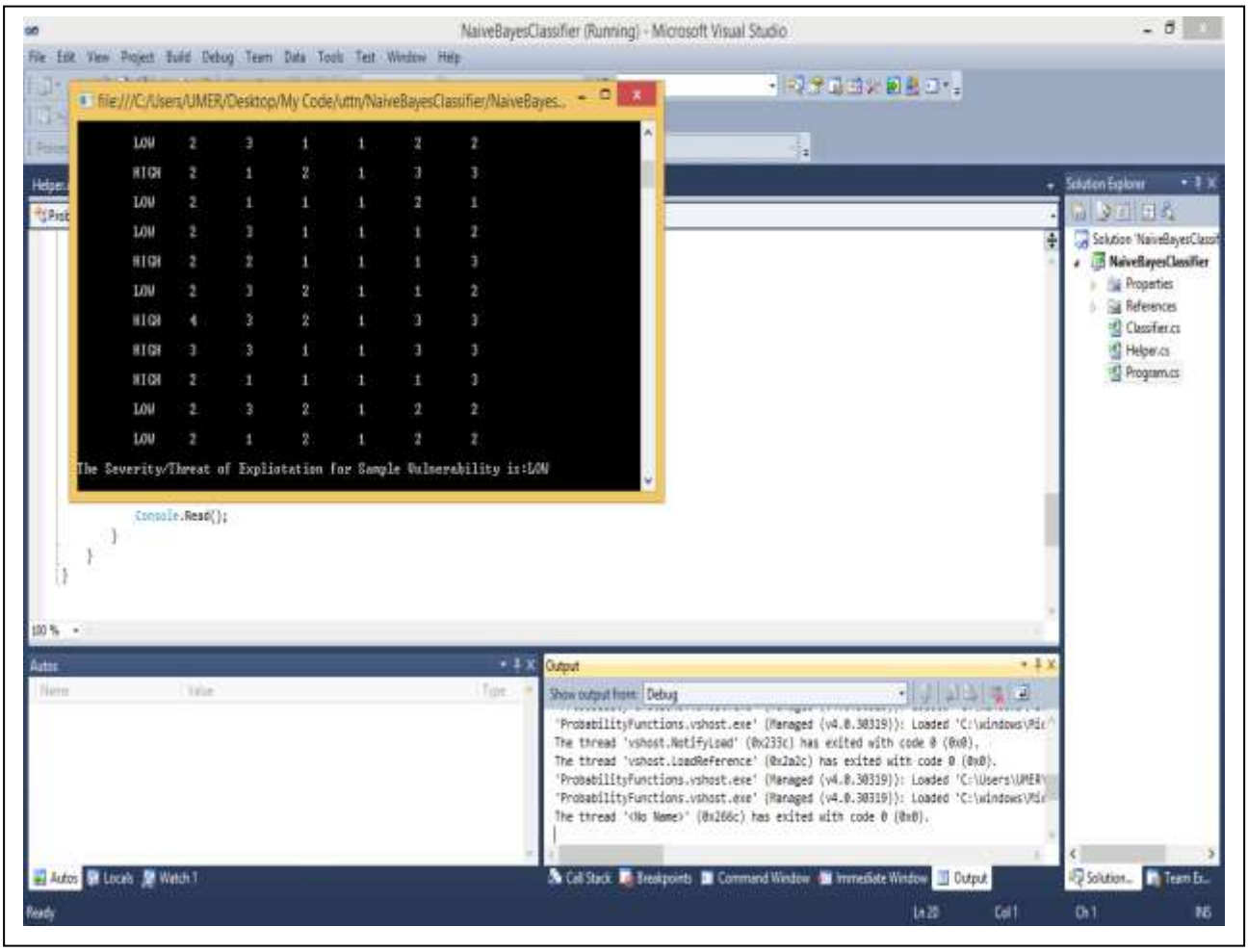
5.2 Exploiting Heartbleed Vulnerability with Metasploit




Metasploit Pro Console
File Edit View Help
[*] 37.187.134.197:443 - Server Hello Session ID: 08f3de2e277
2e18f1410dbc2b8f87a0b787c5079c397bfef191cf38a54ac33e8
[*] 37.187.134.197:443 - SSL record #2:
[*] 37.187.134.197:443 - Type: 22
[*] 37.187.134.197:443 - Version: 0x0301
[*] 37.187.134.197:443 - Length: 1049
[*] 37.187.134.197:443 - Handshake #1:
[*] 37.187.134.197:443 - Length: 1045
[*] 37.187.134.197:443 - Type: Certificate Data (11)
[*] 37.187.134.197:443 - Certificates length: 1042
[*] 37.187.134.197:443 - Data length: 1045
[*] 37.187.134.197:443 - Certificate #1:
[*] 37.187.134.197:443 - Certificate #1: Length: 1039
[*] 37.187.134.197:443 - Certificate #1: #<OpenSSL::X509::Ce
rtificate: subject=#<OpenSSL::X509::Name:0xe8e4140>, issuer=#<OpenSSL::X509::Name:0
xe8e4158>, serial=#<OpenSSL::BN:0xe8e4170>, not_before=2014-03-27 17:04:02 UTC, not
_after=2041-08-11 17:04:02 UTC>
[*] 37.187.134.197:443 - SSL record #3:
[*] 37.187.134.197:443 - Type: 22
[*] 37.187.134.197:443 - Version: 0x0301
[*] 37.187.134.197:443 - Length: 4
[*] 37.187.134.197:443 - Handshake #1:
[*] 37.187.134.197:443 - Length: 0
[*] 37.187.134.197:443 - Type: Server Hello Done (14)
[*] 37.187.134.197:443 - Sending Heartbeat...
Ready
26x83


Metasploit Pro Console
File Edit View Help
mpact";s:3:"120";}s:22:"buttonLayoutLazyHeight";a:2:{s:6:"Normal";s:2:"69";s:7:"Compact";s:2:"2
2";}}s:18:"dd_button_comments";O:11:"DD_Comments":30:{s:19:"islazyLoadAvailable";b:0;s:12:"butt
onLayout";a:1:{s:6:"Normal";s:6:"Normal";}s:4:"name";s:8:"Comments";s:10:"websiteURL";s:11:"htt
p://none";s:6:"apiURL";s:11:"http://none";s:7:"baseURL";s:142:" <a class='
clcount' href=VOTE_URL>COMMENTS_COUNT<a class='clink' href=VOTE
_URL> ";s:12:"baseURL_lazy";N;s:19:"baseURL_lazy_script";N;s:21:"scheduler_lazy_script
";N;s:20:"scheduler_lazy_timer";N;s:8:"finalURL";N;s:13:"finalURL_lazy";N;s:20:"finalURL_lazy_s
cript";N;s:27:"final_scheduler_lazy_script";N;s:16:"isEncodeRequired";b:1;s:10:"wp_options";a:5
:{s:22:"dd_comments_appendType";s:0:"";s:24:"dd_comments_buttonDesign";s:0:"";s:27:"dd_comments
_ajax_left_float";s:0:"";s:21:"dd_comments_lazy_load";s:0:"";s:25:"dd_comments_button_weight";s
:2:"88";}s:18:"option_append_type";s:22:"dd_comments_appendType";s:20:"option_button_design";s:
24:"dd_comments_buttonDesign";s:20:"option_button_weight";s:25:"dd_comments_button_weight";s:22
:"option_ajax_left_float";s:27:"dd_comments_ajax_left_float";s:16:"option_lazy_load";s:21:"dd_c
omments_lazy_load";s:19:"button_weight_value";s:2:"88";s:19:"float_button_design";s:6:"Normal";
s:11:"append_type";s:4:"none";s:13:"button_design";s:6:"Normal";s:15:"ajax_left_float";b:0;s:9:
"lazy_load";b:0;s:16:"buttonLayoutLazy";a:2:{s:6:"Normal";s:6:"Normal";s:7:"Compact";s:7:"Compa
ct";}s:21:"buttonLayoutLazyWidth";a:2:{s:6:"Normal";s:2:"51";s:7:"Compact";s:3:"120";}s:22:"but
tonLayoutLazyHeight";a:2:{s:6:"Normal";s:2:"69";s:7:"Compact";s:2:"22";}}s:18:"dd_button_linked
in";O:11:"DD_Linkedin":30:{s:11:"append_type";s:10:"left_float";s:13:"button_design";s:6:"Norma
l";s:15:"ajax_left_float";s:2:"on";s:9:"lazy_load";b:0;s:12:"buttonLayout";a:3:{s:6:"Normal";s:
3:"top";s:7:"Compact";s:5:"right";s:7:"NoCount";s:4:"none";}s:16:"buttonLayoutLazy";a:3:{s:6:"N
ormal";s:3:"top";s:7:"Compact";s:5:"right";s:7:"NoCount";s:4:"none";}s:16:"isEncodeRequired";b:
0;s:4:"name";s:8:"Linkedin";s:10:"websiteURL";s:23:"http://www.linkedin.com";s:6:"apiURL";s:34:
"http://www.linkedin.com/publishers";s:7:"baseURL";s:163:"<script src='//platform.linkedin.com/
in.js' type='text/javascript'></script><script type='IN/Share' data-url='VOTE_URL' data-counter
Ready
25x95

5.3 Output Of Gaussian Naive Bayes Method for Prediction of Severity of Exploitation for a Sample Vulnerability



5.4 C# Code Segments for Predicting Severity/Threat Of Exploitation using Gaussian Naive Bayes Approach

5.4.1 Populating DataTable & Classifying Data Using Classify Function

```
using System;
using System.Data;

namespace ProbabilityFunctions
{
    public class Program
    {

        static void Main(string[] args)
        {

            // Poplulate Datatable with attributes ...


            DataTable table = new DataTable();
            table.Columns.Add("SEVERITY_OF_EXPLOITATION");
            table.Columns.Add("CVSS_V2_BASE_SCORE");
            table.Columns.Add("ACCESS_VECTOR");
            table.Columns.Add("ACCESS_COMPLEXITY");
            table.Columns.Add("AUTHENTICATION");
            table.Columns.Add("CONFIDIENTIALITY_IMPACT");
            table.Columns.Add("INTEGRITY_IMPACT");
            table.Columns.Add("AVAILABILTY_IMPACT");

            // Read the Complete Dataset based on the above attributes

            //PhpMyAdmin Reflected cross- Site  Scripting Vulnerability(CVE-
2013-1937)
            table.Rows.Add("LOW", 2, 3, 2, 1, 1, 2, 1);
            //MySQL Stored SQL Injection (CVE-2013-0375)
            table.Rows.Add("HIGH", 2, 3, 1, 2, 2, 2, 1);
            //SSL v3 POODLE Vulnerability(CVE_2014-3568)
            table.Rows.Add("LOW", 2, 3, 2, 1, 2, 1, 1);
            //VMWare Guest to Host Escape Vulnerability(CVE-2012-1516)
            table.Rows.Add("HIGH", 4, 3, 1, 2, 3, 3, 3);
            //Apache Tomcat XML Parser Vulnerability(CVE-2009-0783)
            table.Rows.Add("HIGH", 2, 1, 1, 1, 2, 2, 2);
            //Cisco IOS Arbitary Command Execution Vulnerability(CVE-2012-0384)
            table.Rows.Add("HIGH", 3, 3, 2, 2, 3, 3, 3);
            //Apple iWork Denial of Service Vulnerability (CVE-2015-1098)
            table.Rows.Add("LOW", 2, 3, 2, 1, 2, 2, 2);
            //OpenSSL Heartbleed Vulnerability(CVE-2014-0160)
            table.Rows.Add("HIGH", 2, 3, 1, 1, 2, 1, 1);
            //GNU Bourne-Again Shell(Bash) 'ShellShock' Vulnerability(CVE-2014-
6271)
            table.Rows.Add("HIGH", 4, 3, 1, 1, 3, 3, 3);
            //DNS Kaminsky Bug(CVE-2008-1447)
            table.Rows.Add("LOW", 2, 3, 1, 1, 2, 2, 1);
            //Sophos Login ScreenByPass Vulnerability(CVE-2014-2005)
            table.Rows.Add("HIGH", 2, 1, 2, 1, 3, 3, 3);
            //Joomla Directory Traversal Vulnerability(CVE-2010-0467)
            table.Rows.Add("LOW", 2, 1, 1, 1, 2, 1, 1);
```

```
            //Cisco Access Control ByPass Vulnerability(CVE-2012-1342)
            table.Rows.Add("LOW", 2, 3, 1, 1, 1, 2, 1);
            //Juniper Proxy ARP Denial of Service Vulnerability(CVE-2013-6014)
            table.Rows.Add("HIGH", 2, 2, 1, 1, 1, 3, 1);
            //DokuWiki Reflected Cross-Site Scripting Attack(CVE-2014-9253)
            table.Rows.Add("LOW", 2, 3, 2, 1, 1, 2, 1);
            //Adobe Acrobat Buffer Overflow Vulnerability(CVE-2009-0658)
            table.Rows.Add("HIGH", 4, 3, 2, 1, 3, 3, 3);
            //Microsoft Windows Bluetooth Remote Code Execution
Vulnerability(CVE-2011-1265)
            table.Rows.Add("HIGH", 3, 3, 1, 1, 3, 3, 3);
            //Apple ios Security control Bypass vulnerability(CVE-2014-2019)
            table.Rows.Add("HIGH", 2, 1, 1, 1, 1, 3, 1);
            //SearchBlox Cross-Site Request Forgery Vulnerability(CVE-2015-
0970)
            table.Rows.Add("LOW", 2, 3, 2, 1, 2, 2, 2);
            //SSL/TLS MITM Vulnerability( CVE-2014-0224)
            table.Rows.Add("LOW", 2, 1, 2, 1, 2, 2, 2);
            //Google Chrome ByPass Vulnerability(CVE-2012-5376)
            table.Rows.Add("HIGH", 4, 3, 1, 1, 3, 3, 3);

            string strOutput = "";


            for (int i = 0; i < table.Rows.Count - 1; i++)
            {
                for (int j = 0; j < table.Columns.Count - 1; j++)
                {

                    strOutput = strOutput + "\t" + table.Rows[i][j].ToString();


                }

                strOutput = strOutput + "\n\n";
            }

            Console.Write(strOutput);
            Console.ReadKey();


            // Train the Dataset
                    Classifier classifier = new Classifier();
            classifier.TrainClassifier(table);


            // Sample Data set
            Console.WriteLine("The Severity/Threat of Expliotation for Sample
Vulnerability is:"
                + classifier.Classify(new double[] { 1, 1, 2, 2, 2, 1, 1 }));

            Console.Read();
        }
    }
}
```

5.4.2 Calculating Probabilities

```
namespace ProbabilityFunctions
{
    public class Classifier
    {
        private DataSet dataSet = new DataSet();

        public DataSet DataSet
        {
            get { return dataSet; }
            set { dataSet = value; }
        }

        public void TrainClassifier(DataTable table)
        {
            dataSet.Tables.Add(table);

            //table
            DataTable GaussianDistribution = dataSet.Tables.Add("Gaussian");
            GaussianDistribution.Columns.Add(table.Columns[0].ColumnName);

            //columns
            for (int i = 1; i < table.Columns.Count; i++)
            {
                GaussianDistribution.Columns.Add(table.Columns[i].ColumnName +
"Mean");
                GaussianDistribution.Columns.Add(table.Columns[i].ColumnName +
"Variance");
            }



            //calculate data
            var results = (from myRow in table.AsEnumerable()
                           group myRow by
myRow.Field<string>(table.Columns[0].ColumnName) into g
                           select new { Name = g.Key, Count = g.Count()
}).ToList();

            for (int j = 0; j < results.Count; j++)
            {
                DataRow row = GaussianDistribution.Rows.Add();
                row[0] = results[j].Name;

                int a = 1;
                for (int i = 1; i < table.Columns.Count; i++)
                {
                    row[a] = Helper.Mean(SelectRows(table, i,
string.Format("{0} = '{1}'", table.Columns[0].ColumnName, results[j].Name)));
                    row[++a] = Helper.Variance(SelectRows(table, i,
string.Format("{0} = '{1}'", table.Columns[0].ColumnName, results[j].Name)));
                    a++;
                }
            }
```

```
        }

        public string Classify(double[] obj)
        {
            Dictionary<string, double> score = new Dictionary<string,
double>();


            var results = (from myRow in dataSet.Tables[0].AsEnumerable()
                           group myRow by
myRow.Field<string>(dataSet.Tables[0].Columns[0].ColumnName) into g
                           select new { Name = g.Key, Count = g.Count()
}).ToList();

            for (int i = 0; i < results.Count; i++)
            {
                List<double> subScoreList = new List<double>();
                int a = 1, b = 1;


                for (int k = 1; k < dataSet.Tables["Gaussian"].Columns.Count; k
= k + 2)
                {

                    double mean = Convert.ToDouble
(dataSet.Tables["Gaussian"].Rows[i][a].ToString ());
                    double variance = Convert.ToDouble
(dataSet.Tables["Gaussian"].Rows[i][++a].ToString ());
                    double result = Convert.ToDouble (Helper.NormalDist(obj[b -
1], mean, Helper.SquareRoot(variance)));
                    subScoreList.Add(result);
                    a++; b++;
                }

                double finalScore = 0.0;
                for (int z = 0; z < subScoreList.Count; z++)
                {
                    if (finalScore == 0.0)
                    {
                        finalScore = subScoreList[z];
                        continue;
                    }

                    finalScore = finalScore * subScoreList[z];
                }

                score.Add(results[i].Name, finalScore * 0.5);
            }

            double maxOne = score.Max(c => c.Value);
            var name = (from c in score
                        where c.Value == maxOne
                        select c.Key).First();

            return name;
        }
```

```
#region Helper Function

        public IEnumerable<double> SelectRows(DataTable table, int column,
string filter)
        {
            List<double> _doubleList = new List<double>();
            DataRow[] rows = table.Select(filter);

            for (int i = 0; i < rows.Length; i++)
            {
                _doubleList.Add(Convert.ToDouble (rows[i][column]));
            }

            return _doubleList;
        }

        public void Clear()
        {
            dataSet = new DataSet();
        }

        #endregion
    }
}
```

5.4.3 Helper Class

```
public static class Helper
    {
        public static double Variance(this IEnumerable<double> source)
        {
            double avg = source.Average();
            double d = source.Aggregate(0.0, (total, next) => total +=
Math.Pow(next - avg, 2));
            if (d == 0)  d = 1;
            return d / (source.Count() - 1);
        }

        public static double Mean(this IEnumerable<double> source)
        {
            if (source.Count() < 1)
                return 0.0;

            double length = source.Count();
            double sum = source.Sum();
            return sum / length;
        }

        public static double NormalDist(double x, double mean, double
standard_dev)
        {
            double fact = standard_dev * Math.Sqrt(2.0 * Math.PI);
            double expo = (x - mean) * (x - mean) / (2.0 * standard_dev *
standard_dev);
            return Math.Exp(-expo) / fact;
        }
```

```
        public static double NORMDIST(double x, double mean, double
standard_dev, bool cumulative)
        {
            const double parts = 50000.0; //large enough to make the trapzoids
small enough

            double lowBound = 0.0;
            if (cumulative) //do integration: trapezoidal rule used here
            {
                double width = (x - lowBound) / (parts - 1.0);
                double integral = 0.0;
                for (int i = 1; i < parts - 1; i++)
                {
                    integral += 0.5 * width * (NormalDist(lowBound + width * i,
mean, standard_dev) +
                        (NormalDist(lowBound + width * (i + 1), mean,
standard_dev)));
                }
                return integral;
            }
            else //return function value
            {
                return NormalDist(x, mean, standard_dev);
            }
        }

        public static double SquareRoot(double source)
        {

            double dblResult  = Math.Sqrt(source);

            return (dblResult);

        }
    }
}
```

CHAPTER 6

CONCLUSION AND RECOMMENDATIONS

6.1 Conclusions

All Heartbleed-vulnerable systems should immediately upgrade to OpenSSL 1.0.1g. If we are not sure whether an application we want to access is Heartbleed vulnerable or not – we should try any one of the Heartbleed detector tools. No action required, if application that we are using, is not vulnerable. But if the application is vulnerable, wait for it to be patched with OpenSSL 1.0.1g. Once the patch is applied, all the users of such applications should follow the application's release documents from the service providers. Typically, steps to follow once the patch is applied are:

- changing our password
- generating private keys again
- certificate revocation and replacement

An important step is to restart the services that are using OpenSSL (like HTTPS, SMTP etc.). Before accessing any SSL/TLS application such as HTTPS, check to see if the application is vulnerable. Do not access or login to any affected sites. Ensure all such vendors or

enterprises related to your business have applied this security patch. Keep your eyes open on such news of security vulnerabilities.

The Heartbleed bug has shaken the Internet community on its dependency on the open source software. Even though OpenSSL is a very popular library, it was not properly scrutinized. One reason might be because of lack of resources and funds. The organizations and developers using open source software should contribute back to these open source communities in terms of donations, reviewing the code, testing and designing. Amazon, Facebook, Google have recently come forward to donate funds to improve open-source security systems [6].

Naive Bayes Classification enables us to prioritize vulnerabilities for remediation. The type of vulnerabilities which are classified as highly exploitable by the proposed methodology ,can be easily exploited with minimum efforts by the hackers, therefore the particular vulnerability needs headlong attention and should be remediated & fixed as early as possible, to prevent the exploitation of any kind.

6.2 Recommendations

To obtain the fix in your application simply upgrade to OpenSSL 1.0.1g.

If upgrading is not practical, we can rebuild our current version of OpenSSL from source without

TLS Heartbeat support by adding the following compile switch:
-DOPENSSL_NO_HEARTBEATS

This switch ensures that the defected code never gets executed.

An effective vulnerability assessment and remediation program must be able to prevent the exploitation of vulnerabilities by detecting and remediating vulnerabilities in covered devices in a timely fashion. Proactively managing vulnerabilities on covered devices will reduce or eliminate the potential for exploitation and save on the resources otherwise needed to respond to incidents after exploitation has occurred. Information Security and Policy (ISP) provides a centrally managed campus service that campus units can use to comply with this requirement [2].

Reference [10]

Issuing Authority: IJTRE

INTERNATIONAL JOURNAL FOR TECHNOLOGICAL RESEARCH IN ENGINEERING

ISSN (Online): 2347 - 4718

Certificate of Publication

This certificate is awarded to "Mehak Bashir" in recognition of publication of the paper entitled "OPEN SSL HEARTBLEED VULNERABILITY & PREDICTION OF SEVERITY OF EXPLOITATION POSED BY SOME OF THE COMMON TYPES OF VULNERABILITIES, BASED ON COMMON VULNERABILITY SCORING SYSTEM (CVSS), USING NAIVE BAYES CLASSIFICATION ALGORITHM" published in IJTRE (International Journal For Technological Research In Engineering), Volume 4, Issue 9, May- 2017.

Publisher|Editor In Chief

Visit @ www.ijtre.com

Mr. Saumil C Patel

REFERENCES

[1]. Yogesh Joshi, Debabrata Das, Subir Saha, "Mitigating Man in the Middle Attack over Secure Sockets Layer," IEEE, pp 1-5, 2009.

[2]. Eman Salem Alashwali,"Cryptographic Vulnerabilities in Real-Life Web Servers," In Proceedings of the The 3rd International Conference on Communication and Information Technology(ICCIT-2013):Digital Information Management and Security Beirut, pp 6-11, 2013.

[3]. Krishna Kant and Ravishankar Iyer, Prasant Mohapatra, "Architectural Impact of Secure Socket Layer on Internet Servers: A Retrospect," IEEE pp 25-26, 2012.

[4]. Neal Leavitt "Internet Security under Attack: The Undermining of Digital Certificates," IEEE Computer Society, pp 17-20, 2011.

[5]. University of Maryland; cybersecurity experts discover lapses in heartbleed bug fix. (2014). NewsRx Health & Science, Retrieved from http://search.proquest.com.jproxy.lib.ecu.edu/docview/1626397458?accountid=10639

[6]. OpenSSL Team" OpenSSL Project," openssl.org, 2014 [Online]. Available: https://www.openssl.org/ [Accessed: June. 12, 2014]

[7]. CODENOMICON "The Heartbleed Bug," heartbleed.com, 29 April 2014 [Online]. Available: http://heartbleed.com/. [Accessed: June. 12, 2014].

[8]. Rish, Irina (2001). An empirical study of the naive Bayes classifier (PDF). IJCAI Workshop on Empirical Methods in AI.

[9]. "Common Vulnerability Scoring System, V2 Development Update". First.org, Inc. Retrieved November 13, 2015.

[10]. My paper publication links (OpenSSL Heartbleed vulnerability & prediction of severity of exploitation posed by some of the common types of vulnerabilities, based on Common Vulnerability Scoring System (CVSS), using Naive Bayes classification algorithm):

http://www.ijtre.com/search-paper
https://archive.org/details/MehakBashir

Google Scholar link:

http://scholar.google.co.in/scholar?hl=en&q=Mehak+Bashir

Download here:

http://www.ijtre.com/images/scripts/2017040917.pdf
https://archive.org/download/MehakBashir/Mehak_Bashir_Research_Paper.pdf
https://www.google.com/url?sa=t&source=web&rct=j&url=http://ijtre.com/images/scripts/2017040917.pdf

View here:

https://archive.org/stream/MehakBashir/Mehak_Bashir_Research_Paper

Google search:

https://www.google.com/search?tbm=bks&q=OpenSSL+Heartbleed+vulnerability+%26+prediction+of+severity+of+exploitation